OVWDB. These are C interfaces requiring that the daemon also bridge from Java to C. The mapper portion of the daemon is responsible for mapping the entity objects in the SAN Manager composite image into the NetView server. Although in some embodiments of the invention, the daemon does not have a persistent store of the information sent to the Netview, it can have such a store of information to optimize communication.

Communication from NetView to SAN Manager is initiated through the NetView Requester 60, which is an executable launched by the NetView console 52. This executable receives callback requests from NetView and forwards these requests to the Console Request Handler 62.

Communication from NetView to SAN Manager can be performed by the Console Request Handler Application. Although shown as a single block, the launched application performs several distinct functions and may be implemented as separate applications. In some embodiments, all menu operations, such as launching a management application, are performed via the Console Request Handler 62. Additionally, any custom screens or dialogs, such as an administration console, can be part of the Request Handler. The Console Request Handler 62 communicates with the SAN Manager and other services via the NetView daemon. Although the Netview daemon and the Console Request Handler are shown as separate blocks, they are preferably packaged as a single service.

Policy Engine and Action Automation

Illustrated SAN manager 20 detects whether a host 12 has exceeded a utilization threshold of its file system (e.g., as defined by the host operating system, the SAN administrator, or otherwise), and dynamically assigns new LUNs to that host. This function of the SAN manager is herein referred to as storage automation service. As shown in FIGURES 7A and 7B, the SAN manager can include a policy engine 38a that is responsible for carrying out policies relating to assignment of LUNs to hosts based on criteria set by the SAN administrator. In particular, the policy engine is responsible for deciding whether or not to assign LUNs to a host, which LUNs should be assigned and whether or not to issue an alert.

With reference to FIGURE 7B, more generally, the policy engine 38a processes events. In particular, the policy engine maps (event, policy) pairs to an action generator 66 and maps actions received from the action generator 66 to an action handler 68. An automation module 70 provides the association between an event and a policy that applies to that event. The event and policy objects are passed to the policy engine which consults its map to find any action generators that have been registered to handle the given (event, policy) pair.

The automation module 70 includes a set of classes (usually from a single Java package) that provide functionality in the policy engine framework. The following classes are utilized:

IpolicyAutomationControl. Classes that implement this interface initialize the automation modules by creating subscribers and registering action generators and action handlers with the policy engine. This interface can be implemented to create an automation module.

IactionGenerator. Classes that implement this interface also implement a generatActions method by convention. This method can take two parameters. The first is an event class that implements IpolicyEvent and the second is a policy class that implements IPolicy. The generateActions method will evaluate the policy as it applies to the event and will generate action objections as appropriate. The generate action objects will be passed back to the policy engine which will dispatch them to the appropriate action handler.

IactionHandler. Classes that implement this interface also implement a handle action method which takes as its sole parameter a class which implements IpoliyAction. The action handler will execute the appropriate measures for the given action.

IPolicy, IPolicyEvent, IpolicyAction. Classes that implement these interfaces wrap information that the action generators and action handlers need in order to perform their functions.

During startup, the policy engine 38a reads a list of classes from its preferences. Each class implements *IpolicyAutomationControl* and represents an automation module. The policy engine will create an instance of each class and call its *initialize()* method, which is responsible for registering action generators and action handlers. In addition, the *initialize()* method can also